**Maven:**

* Apache Maven is Software Project Management and Build Management tool for JAVA Framework (so, it can be Java Projects, Selenium automation code in Java, Rest Assured automation using Java or Mobile automation code in Appium-Java)
* Why Maven:
  + Central Repository to get Dependencies:

Maven maintain all dependency files / jars which are required in Java framework building. Using just few line of xml code in pom.xml, required dependencies can be auto-downloaded in a project folder. You don’t need to explicitly download and maintain those jar files.

* + Maintain common structure across the organization:

Maven has many project folder templates. So while creating a project, you can choose template according to your project or organization so that appropriate folder structure will get created. We don’t need to create any different folder structure our own; because of these consistency can be maintain across the bigger projects or organization

* + Flexibility in integration with CI tool:

It supports and easy to integrate with CI tool, Jenkin.

* + Plugins for Test Framework execution:

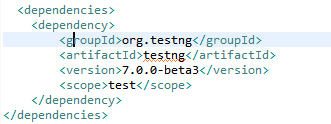
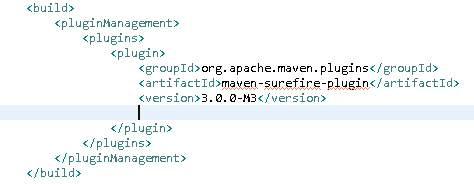
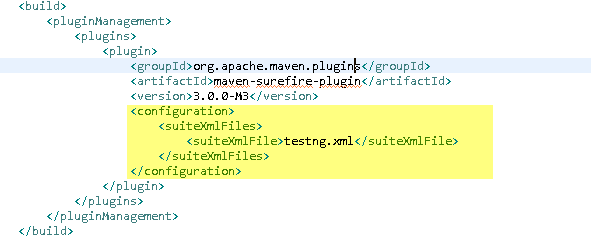
It supports different Test frameworks like TestNG, JUnit. And it supports different plugins accordingly.

* Generally Maven gets installed with newer version of eClipse; otherwise we can add Maven into eClispe thru eClipse> Help >eClipseMarketPlace feature
* Same, you can add TestNG into eClipse
* Also, you have to download a ready-made binary distribution archival file from <https://maven.apache.org/download.cgi> and follow installation instructions given <https://maven.apache.org/install.html>
* Unzip the downloaded file in your dedicated folder
* Ensure JAVA\_HOME environment variable is set and points to your JDK installation
* Add the bin directory of the created directory apache-maven-3.6.0 to the PATH environment variable

C:\Gopal P\eClipse\eclipse\apache-maven-3.5.4\bin

* Create new MAVEN\_HOME environment variable and set the path before bin

C:\Gopal P\eClipse\eclipse\apache-maven-3.5.4

* Just to check you can run the command c:\>mvn --version
* You will get the maven version
* You can also see ‘Maven’ option in eClipse> Windows > Preferences
* You can find Local repositories path at eClipse>Windows >Preferences > Maven > User Settings > Local Repository
* When Maven program runs, it checks all required dependency files under local repository and use them for execution; otherwise, it will download from Maven Repositories to this local folder.
* We need to add dependencies in pom.xml which will pull required jars from maven repository
* <groupId>:groupId will identify project uniquely across all the projects. It is the project folder or main path where maven look for that dependency file at it’s server to get downloaded. Also, it’s a Package Name
* <artifactId>:it is a file, usually a JAR, under the main folder/path where maven look for that dependency file at it’s server to get downloaded. Also, it’s Project Name.
* When you go in Eclipse Java Project and menu ‘Windows’ > Preferences; you should see ‘Maven’ in the list; meaning this Eclipse version supports Maven.
* When you goto‘Windows’ > Preferences> Maven > User Settings, you can see Local Repository path. It is the path/place of your local machine where JARs will get downloaded by default.
* Add dependencies under <dependencies> tag
* 
* Test Class name (under src/test/java) should include “Test” keyword at the end of their name
* Surefire plugin identify the executable tests by the keyword “Test” and execute them
* We have to add Surefire plugin in pom.xml along with other dependencies.
* 
* When you run the command from Project Folder :/>mvn test , it will execute all the test classes which are in src/test/java folder
* If we have to execute TestNG.xml thru maven, then we need to add such configuration in Pom.xml <https://maven.apache.org/surefire/maven-surefire-plugin/examples/testng.html>
* 
* When :/>mvn test command will run, it will execute the testing.xml
* If there is any version error of compiler source of compiler target, we can add following two lines in pom.xml under <Properties>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

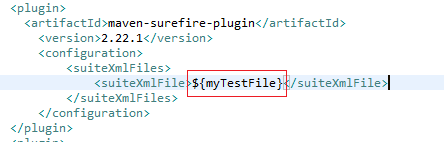
<maven.compiler.source>1.6</maven.compiler.source>

<maven.compiler.target>1.6</maven.compiler.target>

</properties>

* If there are multiple testng.xml created; let’s say one for Regression and other for Smoke Test and based on requirement, we have to execute specific testng.xml then we can create different **Profiles** in pom.xml and execute it accordingly.
* 
* So if you want to run only Smoke test, you can execute this command :/>mvn test –PSmoke

Where ‘–P’ stands for profile and ‘Smoke’ is profile name

* If you don’t want to hardcode the ‘testng.xml’ file name into the POM.xml then you can use variable there as below and provide the variable value in command prompt or Jenkins.
* 
* :/> mvn clean test **-DmyTestFile**=**testng\_Smoke.xml**
* If you have to run single test class, then use following command

:/>mvn –Dtest=MobileTest test

Where, ‘MobileTest’ is a class name which will be executed.

* Note that, ‘Test’ word should be there is classname so that maven willpickup that class for execution. E.g. MyClassTest.java
* :\>mvn clean 🡺 to clean the previous builds/references/temporary files
* :\>mvn compile => to compile the mvn project & check for syntax errors if any
* :\>mvn test => to execute the mvn project/class. Program gets compiled automatically when you run the program
* When you run these above commands, you have to run it from project folder. E.g. C:\my test\myproject:\>mvn test
* GroupID: GroupId uniquely identifies your project across all projects. A group ID should follow Java's package name rules. This means it starts with a reversed domain name you control.if the current project is a multiple module project, it should append a new identifier to the parent's groupId. E.g. org.apache.maven, org.apache.maven.plugins, org.apache.maven.reporting
* ArtifactID:ArtifactID is the name of JAR.
* ArcheTypeArtifact: Archetype is Maven project templating toolkit. Archetype helps to create a maven project template with certain project folder structures, parameters etc. e.g. ‘maven-archetype-quickstart’ archetype creates sample maven project where it creates project folder including ‘src’ subfolder and pom.xml with certain parameters. ‘src’ folder contains ‘main’ and ‘test’ subfolders.
* Surefire plugin: It used during test phase. It executes tests which are present in src/test/java folder. Maven Surefire plugin is used to follow sequence of the test in testng.xml file. If we don’t include it in pom.xml, then it will execute all the test cases which has prefix or suffix as ‘test’ and these test will execute without any sequence.